Carrera: Desarrollo de Software
Semestre 05

Programa de la asignatura:
Programación orientada a objetos III

Unidad 1. Archivos

Clave:
16142524

Universidad Abierta y a Distancia de México

Índice

Presentación de la unidad 3
Propósitos de la unidad 3
Competencia específica 3
Temario de la unidad 3
Tema 1. Concepto de flujo de e/s 4
Tema 2. Manejo de archivos 6
Cierre de la unidad 12
Para saber más… 13
Fuentes de consulta 13

Presentación de la unidad

Al crear programas y ejecutarlos la información que se les proporciona permanece en la memoria del ordenador y, al cerrar el programa, dicha información se pierde, pero no siempre basta con realizar una ejecución, en muchos casos es necesario almacenar información de manera permanente, una de las formas de realizar este almacenamiento es la manipulación de archivos.

En esta primera unidad de la materia Programación orientada a objetos III (POO3), veremos el flujo de entrada y salida de información de un programa, así como el almacenamiento de información mediante la manipulación de archivos. Los temas que serán tratados son fundamentales por la importancia que tiene aprender a almacenar la información que un programa puede llegar a producir; una forma de almacenar es mediante los archivos, otra, es mediante el uso de bases de datos, tema que ya se ha tratado previamente en POO2.

Propósitos de la unidad

En esta unidad lograrás:

- Identificar los flujos de entrada y salida de un programa
- Identificar las clases que permiten manejar flujos de entrada y salida de un programa.
- Manipular archivos.

Competencia específica

Diseñar clases para manipular datos mediante las operaciones básicas de los archivos.

Temario de la unidad

1. Archivos

1.1. Concepto de flujos de E/S
- 1.1.1. Clases que permiten manejar flujos de entrada
- 1.1.2. Clases que permiten manejar flujos de salida

1.2. Manejo de archivos
- 1.2.1. Creación de un archivo
- 1.2.2. Lectura y escritura de un archivo
- 1.2.3. Eliminación y renombramiento de archivos

Tema 1. Concepto de flujo de E/S

Un programa manipula la información dentro de sí mismo y mantiene los datos que produce en memoria mientras está en ejecución, pero estos datos se pierden cuando se termina la ejecución del programa. Si se pretende mostrar u obtener alguna información de un lugar externo es necesario que el programa cree un flujo de datos para poder manejarlos (algo parecido al canal de comunicación que se abre para conectarse a una base de datos). Por lo tanto hay un flujo de entrada que recibe los datos desde el exterior del programa y un flujo de salida que envía los datos hacia el ordenador u otro dispositivo de almacenamiento.

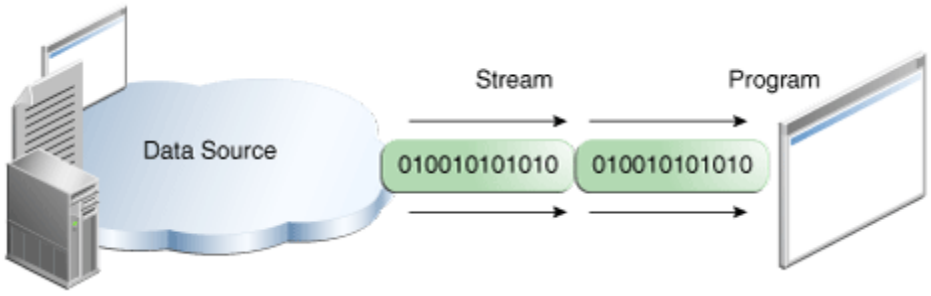
Para la entrada y salida de datos de un programa es necesario manejar los flujos de información que se generan entre el programa y el quipo donde se está ejecutando, a estos flujos se les conoce como *streams*.

Un *stream*, cuya traducción literal es “flujo”, es una abstracción de todo aquello que produce o consume información. La vinculación de este *stream* al dispositivo físico la hace el sistema de entrada y salida de Java (Suarez, 2001).

Para el manejo flujos de entrada/salida, Java nos proporciona dos tipos de flujos posibles, los de *byte* y los de caracteres. Los cuales se describen a continuación:

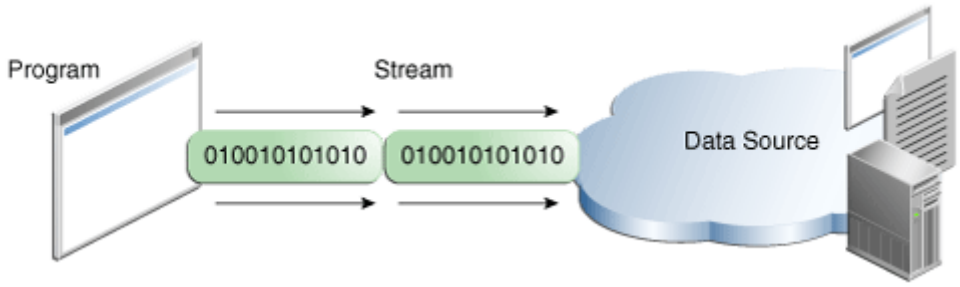
- Flujo de bytes (*ByteStreams*): Nos proporciona un medio adecuado para el manejo de entradas y salidas de *bytes;* su uso lógicamente está orientado a la lectura y escritura de datos binarios (Suarez, 2001).
- Flujo de caracteres (*CharacterStreams*): Proporciona un medio adecuado para el manejo de entradas y salidas de caracteres (Suarez, 2001).

El manejo de flujos se divide en los flujos que permiten la entrada de datos del exterior hacia un programa, tal como se ilustra en la siguiente imagen.



(Imágenes obtenidas de Oracle, 2012)

Y los flujos que permiten la salida de información del programa hacia el exterior.



*(*Imágenes obtenida de Oracle, 2012*)*

Para conocer más a fondo las clases que controlan la entrada y salida de datos de un programa revisa los siguientes textos[1]:

- En tema 9.1 Clases de Java para lectura y escritura de datos. De *Aprenda java como si estuviera en primero*, en García (2000, p. 151); en este texto, se muestra una descripción de cada una de las clases de entrada/salida de datos, es importante que **leas** este material para que comprendas para qué sirve cada una de estas clases y puedas hacer uso de ellas en cada problemática que se te presente.
- Con respecto a "Entrada y salida", en Sánchez (2004, p. 93) se presenta una descripción de las clases para dar facilidades de entrada y salida, se muestra también una explicación sobre las clases abstractas que definen las funciones básicas de lectura y escritura. En este mismo capítulo se muestra un agrupamiento de las clases que manejan la entrada/salida de datos.
- En la siguiente liga http://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html de Oracle (2012) encontrarás la descripción del uso de flujos de *bytes*, los cuales son utilizados para cuando la entrada/salida de datos consta de datos representados mediante *bytes*.
- En la siguiente liga: http://docs.oracle.com/javase/tutorial/essential/io/charstreams.html de Oracle (2012) encontrarás la descripción del uso de flujos basados en caracteres.

Revisa cada uno de los textos señalados, pues, estas clases permiten la entrada y salida de datos, pero al utilizarlas de manera directa van entrando y saliendo dato por dato, lo que hace que el rendimiento de programas ejecutado de esta manera sea muy pobre. Para mejorar el rendimiento Java proporciona clases llamadas *buffers*.

Para García (2000), un "buffer es un espacio de memoria intermedia que actúa de "colchón" de datos. Cuando se necesita un dato del disco se trae a memoria ese dato y sus datos contiguos, de modo que la siguiente vez que se necesite algo del disco la probabilidad de que esté ya en memoria, sea muy alta. Algo similar se hace en la escritura, intentando

[1] Para profundizar en el tema y cumplir con las tareas de la unidad, ingresa a la sección de materiales de apoyo de la unidad. Consulta los textos de la unidad que se agregan en formato PDF

realizar en una sola operación de escritura física varias sentencias individuales de escritura" (p. 160).

Para adentrarte en el manejo de *buffers* consulta el tutorial de Java que encontrarás en la siguiente liga, http://docs.oracle.com/javase/tutorial/essential/io/buffers.html. En éste, encontrarás una descripción de lo que son los *buffers* así como de la sintaxis requerida para su uso.

También puedes consultar el tema 9.1.3 Clases que añaden características. En *Aprenda java como si estuviera en primero*, en García (2000, p. 154) encontrarás una descripción detallada de las clases que ayudan con el manejo de flujos, entre ellas se encuentran los *buffers*.

En este primer tema de la unidad, revisaste los flujos de datos y distinguiste entre los flujos de entrada y los de salida, también abordaste el tema de *buffers* que ayudan con el manejo de flujos de datos al momento de leer o escribir en un archivo, así mismo analizaste su utilidad en el manejo de archivos. Lo más importante es que identifiques las clases que se requieren para el manejo de estos flujos, las cuales estarás utilizando en el siguiente tema.

Tema 2. Manejo de archivos

En el tema anterior viste que, para manejar archivos, es importante primero manejar los flujos de datos para permitir la comunicación con información externa al programa; en el presente tema estudiarás la manipulación de archivos, cómo leer y escribir en archivos, así como la creación de nuevos archivos, eliminación y renombramiento de éstos.

Los **archivos de datos** son un conjunto de datos estructurados, que se tratan como una unidad y se encuentran almacenados en algún dispositivo creado para este fin (disco duro o memoria flash, por ejemplo) (Joyanes, 2002). Los archivos son tomados por Java como una sola cosa, dentro de la cual pueden existir numerosas formas diferentes (caracteres, dígitos, imágenes, etc.).

Para manejar los archivos utilizarás la clase *File, la cual* no trabaja sobre un flujo, sino que trata directamente con el fichero y con el sistema de ficheros. Con esta clase no accedes a los datos, pues está orientada a obtener y/o manipular la información asociada al archivo, por ejemplo: permisos, fechas, saber si es un fichero o un directorio, etc. Así, cuando crees un objeto de esta clase, lo que estás haciendo es establecer un enlace con un archivo o un directorio físico, al que luego le podrás consultar y/o afectar sus propiedades (*Suarez, 2001*). Esta clase cuenta con los métodos *delete* y *renameTo*, para borrar y renombrar archivos respectivamente (Joyanes, 2002).

Para poder realizar la manipulación de archivos entonces deberemos hacer uso de todos los elementos mencionados, por ejemplo si vamos a crear un archivo se necesita:

- Crear un objeto de la clase *File*, y a este objeto hay que enviarle la dirección y nombre del archivo que va a crear (es decir dónde se va a ubicar).
- El manejo del flujo de salida (pues los datos *saldrán* del programa hacia el archivo). Esto puede ser con o sin *buffer*.
- Una vez que se tengan los elementos anteriores sólo hace falta invocar el método de escritura.

Para analizar a fondo este proceso, **consulta** los siguientes materiales[2]:

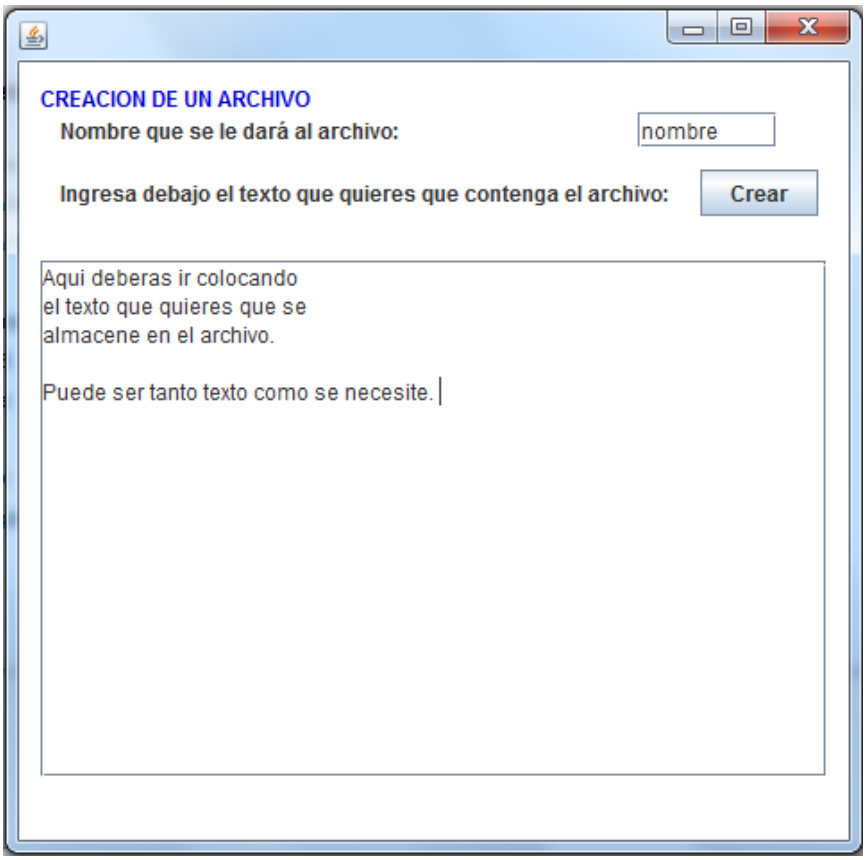
- En Froufe (2009, p. 159) podrás analizar la clase *file*, la cual te ayudará para la manipulación de archivos, a lo largo de todo ese capítulo se describe el manejo de archivos y las clases que te ayudan a realizarlo.
- Con respecto al tema de Ficheros, en Sánchez (2004, p. 99) podrás ver cómo escribir en un archivo, así como manejar las rutas de los archivos que se vayan a crear.
- En el tema de Combinación de las clases, en *Suarez (2001, p. 10*), podrás ver algunos ejemplos que combinan las clases que tratan los flujos de entrada/salida; es importante que revises a detalle los ejemplos que se presentan, posterior a ello, es importante que los codifiques tú mismo/a para que comprendas su uso.
- Con respecto a la clase *File I/O*, en Oracle (2011) disponible en la siguiente liga: http://docs.oracle.com/javase/tutorial/essential/io/fileio.html revisa la forma en que se manipulan los archivos.
- En el *capítulo 6 Acceso al disco* de Martin (2010, p. 278) encontrarás como almacenar información en ficheros del disco duro, así como su recuperación.

Después de que revisaste los textos referente a los flujos de información de entrada y salida (I/O) y que abordaste el tema de la clase *File,* que es la que Java ha proporcionado para el manejo de archivos, a continuación revisarás ejemplos de cómo se utilizan estas clases; es recomendable que realices tus propios programas basándote en la sintaxis que se muestra en los ejemplos, todo ello para que analices a profundidad el código y logres aplicarlo en las actividades que realizarás para este tema. Revisa los siguientes códigos que se te presentan para que logres realizar tus actividades con mayor facilidad.

* Nota: Te presentaremos sólo los métodos que realizarán el manejo de archivos, la interfaz gráfica que manejará la información será omitida, pues el código para realizarlo ya se revisó en POO2.

Creación de archivos:
El primer ejemplo es para crear un archivo, la interfaz gráfica deberá ser similar a la que se muestra a continuación:

[2] Ingresa a la sección de materiales de apoyo de la unidad, para profundizar en cada uno de los temas.


CREACION DE UN ARCHIVO
Nombre que se le dará al archivo:
nombre
Ingresa debajo el texto que quieres que contenga el archivo:
Crear
Aqui deberas ir colocando
el texto que quieres que se
almacene en el archivo.
Puede ser tanto texto como se necesite.

A continuación se presentan instrucciones y un ejemplo de la creación de archivo:

Ejemplo de código 1. Crear un archivo

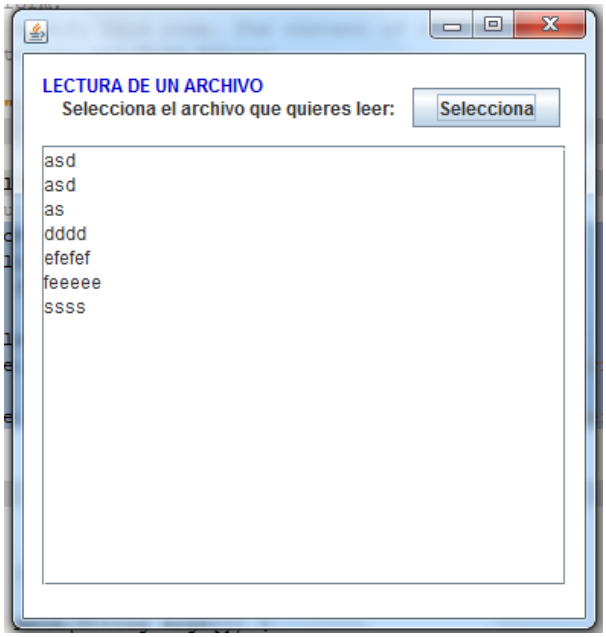
```
private void btnCrearActionPerformed(java.awt.event.ActionEvent evt) {
    String nombreArchivo=txtNombre.getText();
    String carpeta = System.getProperty("user.dir");
    String direccionCompleta= carpeta+"/"+nombreArchivo+".txt";
    FileWriter ubicacion = null;
    try {
        ubicacion = new FileWriter(direccionCompleta);
    } catch (IOException ex) {
        Logger.getLogger(crear.class.getName()).log(Level.SEVERE, null, ex);
    }
    try{
        BufferedWriter escritor = new BufferedWriter(ubicacion);
        escritor.write(txaArchivo.getText());
        escritor.close();
    } catch(Exception e){
    }
}
```

El ejemplo se presenta de la siguiente manera:

- Lo primero que necesitamos es saber que nombre se le dará al archivo, esto se obtiene de una caja de texto donde el usuario captura el nombre.
- En seguida, se debe obtener el directorio donde se ubicará el archivo, en este caso el archivo será almacenado en la misma dirección donde se encuentra el proyecto, obtenemos esa dirección mediante System.getProperty("user.dir").
- Ya con el nombre y la dirección, creamos la dirección completa de la ubicación y el nombre que tendrá el archivo.
- Después, en morado, se muestra la creación de un *FileWriter* que se crea en base a la dirección del archivo que se armó en la sección anterior (azul). Nótese que la creación del objeto de *FileWriter* está en un bloque try-catch, porque la creación de ese objeto es susceptible a errores si no se arma correctamente el *path* que tendrá el archivo.
- En seguida, se crea un *buffer* que transportará la información que el usuario haya capturado en el área de texto (naranja).
- Por último en verde se cierra el *buffer*.

Lectura de un archivo:

El siguiente ejemplo es para leer un archivo, la interfaz gráfica que obtendrás deberá ser similar a la que se muestra a continuación:



Para obtenerla tendrás que hacer lo siguiente:

- Primero, se crea un objeto de la clase *JFileChooser*, esta clase se reviso en POO2, recuerda que genera una ventana gráfica para explorar los archivos de la computadora en la que se ejecuta el programa. Y ya que se abrió el diálogo, se almacena en el objeto de tipo *File* (se muestra en azul en el ejemplo de código 2).
- Después, se crea un objeto de tipo *FileReader* (que como su nombre lo indica lee archivos), se muestra en morado.

- A continuación, se crea un objeto de tipo *buffer* de lectura (*BufferedReader*), se muestra en verde.
- En rojo, se muestra como sacar los datos del *buffer*; para ello se va sacando línea por línea, pues no podemos hacer una extracción de todo el archivo en un sólo paso, esto lo vamos concatenando en la variable texto.
- Por último se muestra ese texto en un área de texto de la interfaz.

Ejemplo de código 2. Lectura de un archivo

```
private void btnSelActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fc=new JFileChooser();
    fc.showOpenDialog(null);
    File archivo = fc.getSelectedFile();
    try{
        FileReader fr = new FileReader (archivo);
        BufferedReader br = new BufferedReader(fr);
        String texto="";
        String linea="";
        while(((linea=br.readLine())!=null) ){
            texto+=linea+"\n";
            }
           txaArchivo.setText(texto);
        }
      catch(Exception e){
    }
}
```

Renombrar un archivo:

Otra función que revisarás es el renombramiento de un archivo: Para ello, primero deberás seleccionar el archivo y después cambiarle el nombre, por lo tanto, deberá crearse una apariencia gráfica similar a la que se muestra a continuación, más adelante verás el código requerido para que los botones de *Selecciona y Cambiar* funcionen correctamente.

.

Ejemplo de código 3. Renombrado de un archivo

```
private void btnSelActionPerformed(java.awt.event.ActionEvent evt) {
```

```
        // TODO add your handling code here:
        JFileChooser fc=new JFileChooser();
        fc.showOpenDialog(null);
        file = fc.getSelectedFile();
}
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        String nvo=file.getParent()+"\\"+txtNvoNombre.getText()+".txt";

        File f2 = new File(txtNvoNombre.getText());
        boolean correcto = file.renameTo(f2);
        if (correcto==true)
            JOptionPane.showMessageDialog(null,"El renombrado ha sido correcto");
        else
            JOptionPane.showMessageDialog(null,"El renombrado no se ha podido realizar");
    }
```

Veamos el código:
El bloque de código en rosa corresponde al método invocado por el evento, al presionar el botón *Selecciona*, como su nombre lo indica, este botón deberá permitir seleccionar el archivo que se quiere renombrar, para ello:

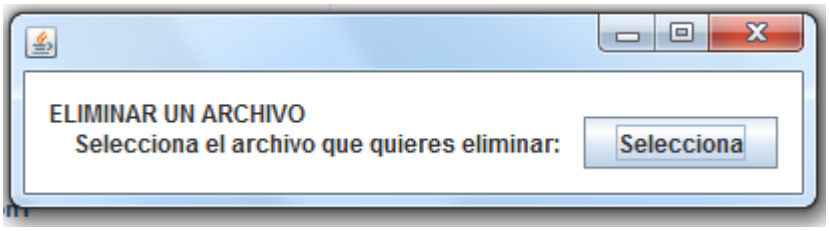
- Primero se crea un objeto de *JFileChooser*.
- Después se abre el diálogo del *JFileChooser*.
- Por último se obtiene el archivo seleccionado y se almacena en una variable de tipo *file*, como puedes darte cuenta esta variable no está declarada en este método, la idea es declarar la variable de tipo *file* de manera global para poder retomar el valor en el método que renombrará el archivo.

Por su parte, el código en azul muestra el método invocado por el evento al presionar el botón *Cambiar.* Lo primero que se debe hacer es crear la nueva ruta y nombre del archivo, deberás seguir los para ello:

- Del archivo elegido para renombrar, se obtiene su directorio (*file.getParent*()).
- Al directorio se le concatena el nuevo nombre que se le quiere dar al archivo, tomando el valor que el usuario introdujo (*txtNvoNombre.getText()).*
- Para completar el nombre hay que concatenar también la extensión que tendrá el archivo.
- Después se crear otra variable tipo *file*, con el nuevo nombre.
- En seguida se renombra el archivo.

Por último, para informar al usuario, se detecta si el archivo se renombró o no, esto se almacena en la variable *correcto*, y se envía un mensaje al usuario para informarlo.

Eliminar un archivo:
Para la eliminación de un archivo sólo necesitamos saber qué archivo es el que se eliminará, es decir que lo seleccionen. Por lo tanto, la apariencia gráfica solo requiere del botón para seleccionar el archivo, tal como se muestra en la siguiente imagen.



Ahora, revisa el código:

- El bloque marcado en rosa es para crear el *JFileChooser*, abrir el diálogo y almacenar el archivo seleccionado.
- El bloque azul elimina el archivo e informa al usuario si ha sido borrado o no.

Ejemplo de código 4. Eliminar un archivo

```
JFileChooser fc=new JFileChooser();
fc.showOpenDialog(null);
File archivo = fc.getSelectedFile();
 if (archivo.delete())
   JOptionPane.showMessageDialog(null, "El fichero ha sido borrado satisfactoriamente");
else
   JOptionPane.showMessageDialog(null, "El fichero no puede ser borrado");
```

Date cuenta que al renombrar y eliminar archivos requieres tener los archivos cerrados, por ello es que no se muestran en la apariencia gráfica.

Por otra parte también puedes ver que para hacer estas operaciones no requieres de ningún buffer, pues no estás cargando ningún dato, sólo estás operando sobre los archivos.

Cierre de la unidad

Has concluido la primera unidad del curso. A lo largo de ésta te has introducido en el concepto de flujos de entrada y salida de un programa; has estudiado las clases que apoyan el manejo de estos flujos, lo que te permitirá distinguir los flujos de datos de entrada/salida, así como la importancia del uso de buffers. También has estudiado la manipulación de archivos y cómo crear un programa, mediante el cual se ponga en práctica los conocimientos adquiridos.

Es aconsejable que revises nuevamente la unidad en caso de que los temas que se acaban de mencionar no te sean familiares o no los recuerdes, de no ser este tu caso, ya estás preparado(a) para seguir con la *Unidad 2. Hilos*, en donde continuarás con la revisión del uso de hilos para la creación de programas multi-flujo.

Para saber más…

Para que puedas ejecutar los programas que se te presentan, así como las actividades es importante que instales un IDE en tu computadora, se recomienda NetBeans, puedes descargarlo de forma gratuita de la siguiente liga: http://netbeans.org/downloads/

Es recomendable que pruebes los códigos que se presentan en los ejemplos que se encuentran en cada fuente de consulta mencionada.

Fuentes de consulta

- Froufe, A. (2009) *Java 2 Manual de usuario y tutorial.* (5ta Ed.). México: Alfaomega.
- García, J. et all. (2000) *Aprenda Java como si estuviera en primero*. España: Tecnun.
- Joyanes, L. (2002) *Java 2 Manual de programación*. México: MC Graw Hill
- Martín, A. (2010) *Programador certificado Java 2 curso práctico 3ra edición*. México: Alfaomega.
- Oracle (2012) *The JAVA Tutorials*. Estados Unidos de América: Oracle. Disponible en: http://docs.oracle.com/javase/tutorial/
- Sanchez, J. (2004) *Java 2*. España: S/E
- Suarez, L. (2001) *El paquete java.io. Javahispano*. Disponible en: http://www.javahispano.com